

Программирование Python

Кафедра ИВТ и ПМ
ЗабГУ

2018

План

Предыдущие темы

IDE

Jupyter

Wing

Отладка

Циклы

while

вычисление сумм и произведений

for

Операторы управления циклом

Составные типы данных

Списки

Outline

Предыдущие темы

IDE

Jupyter

Wing

Отладка

Циклы

while

вычисление сумм и произведений

for

Операторы управления циклом

Составные типы данных

Списки

Предыдущие темы

- ▶ Python компилируемый или интерпретируемый язык?
- ▶ Как объявить переменную?

Предыдущие темы

- ▶ Python компилируемый или интерпретируемый язык?
- ▶ Как объявить переменную?
`myvariable = 123`
- ▶ Как определить тип переменной?

Предыдущие темы

- ▶ Python компилируемый или интерпретируемый язык?
- ▶ Как объявить переменную?
`myvariable = 123`
- ▶ Как определить тип переменной?
По типу заданного ей значения

Предыдущие темы

- ▶ Python компилируемый или интерпретируемый язык?
- ▶ Как объявить переменную?
`myvariable = 123`
- ▶ Как определить тип переменной?
По типу заданного ей значения
вызвав функцию: `type (имя-переменной)`
- ▶ Назовите основные простые типы

Предыдущие темы

- ▶ Python компилируемый или интерпретируемый язык?
- ▶ Как объявить переменную?
`myvariable = 123`
- ▶ Как определить тип переменной?
По типу заданного ей значения
вызвав функцию: `type (имя-переменной)`
- ▶ Назовите основные простые типы
`bool` - логический тип
`int` - целое число
`float` - вещественное число
- ▶ Как подключить пакет (модуль)?

Предыдущие темы

- ▶ Python компилируемый или интерпретируемый язык?
- ▶ Как объявить переменную?

```
myvariable = 123
```

- ▶ Как определить тип переменной?
По типу заданного ей значения
вызвав функцию: `type (имя-переменной)`

- ▶ Назовите основные простые типы

`bool` - логический тип

`int` - целое число

`float` - вещественное число

- ▶ Как подключить пакет (модуль)?

```
import math
```

```
import math as M
```

```
from math import *
```

Предыдущие темы

- ▶ Как выглядит условный оператор?

Предыдущие темы

- ▶ Как выглядит условный оператор?

```
if логическое-выражение :  
    оператор1  
    оператор2  
    ...  
else:  
    операторN  
    операторN1  
    ...
```

Предыдущие темы

- ▶ Как выглядит условный оператор?

```
if логическое-выражение :  
    оператор1  
    оператор2  
    ...  
else:  
    операторN  
    операторN1  
    ...
```

- ▶ Зачем нужны отступы?
- ▶ Где заканчивается условный оператор?
- ▶ Сколько отступов рекомендуется использовать для одного блока?

Предыдущие темы

- ▶ Как вывести что-то на экран?

Предыдущие темы

- ▶ Как вывести что-то на экран?

```
print("текст")  
print("скорость равна = ", v, "м/с")
```

- ▶ Как прочитать данные с клавиатуры?

Предыдущие темы

- ▶ Как вывести что-то на экран?

```
print("текст")  
print("скорость равна = ", v, "м/с")
```

- ▶ Как прочитать данные с клавиатуры?

```
x = float( input("Введите X ") )  
ans = input("Для продолжения введите Y или Yes: ")
```

Outline

Предыдущие темы

IDE

Jupyter

Wing

Отладка

Циклы

while

вычисление сумм и произведений

for

Операторы управления циклом

Составные типы данных

Списки

Outline

Предыдущие темы

IDE

Jupyter

Wing

Отладка

Циклы

while

вычисление сумм и произведений

for

Операторы управления циклом

Составные типы данных

Списки

Горячие клавиши

Ctrl + Enter - Выполнить код в текущей ячейке

Ctrl + Shift - Выполнить код в текущей ячейке, перейти на следующую (если следующей нет, то она создаётся)

Alt + Enter - Вставить ячейку после текущей

- ▶ Весь код и текст содержится в так называемых *блокнотах* (notebooks)
- ▶ Блокноты сохраняются автоматически при создании (скорее всего в каталоге текущего пользователя)
- ▶ Файл блокнота имеет расширение **ipynb**
- ▶ Jupyter блокнот - это XML файл который, помимо прочего может содержать изображения.

- ▶ Блокноты состоят из яйчеек
- ▶ Яйчейки бывают разные:
 - ▶ Яйчейки с текстом. (Markdown)
Можно использовать LaTeX для описания формул
 - ▶ Яйчейки с кодом (code)

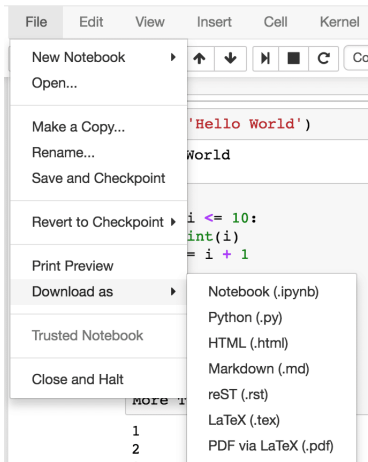
Доклад: Как использовать Jupyter (ipython-notebook) на 100%

Jupyter

- Сохранение notebook.

Чтобы иметь возможность запускать код блокнота в других средах программирования нужно сохранить его как **py** файл.

Меню File -> Download as -> Python (.py)



- ▶ Открытие py файлов в Jupyter.

Jupyter не работает непосредственно с py файлами, но их можно загружать в блокнот с помощью команды `%load`:

```
%load my_genious_program.py
```

Файл должен быть расположен в том же каталоге, что и блокнот Jupyter иначе нужно указать полный или относительный путь до него.

Текст загруженной программы будет помещён в текущую ячейку.

- ▶ Напечатать путь к текущей папке

```
%pwd
```

Демонстрация

Outline

Предыдущие темы

IDE

Jupyter

Wing

Отладка

Циклы

while

вычисление сумм и произведений

for

Операторы управления циклом

Составные типы данных

Списки

Демонстрация

Настройки → Файлы → кодировка по-умолчанию: UTF-8.

Ошибки - во вкладке Exceptions

Ввод и вывод во вкладке Debug IO

Outline

Предыдущие темы

IDE

Jupyter

Wing

Отладка

Циклы

while

вычисление сумм и произведений

for

Операторы управления циклом

Составные типы данных

Списки

Отладка — этап разработки компьютерной программы, на котором обнаруживают, локализуют и устраняют ошибки.

¹трассировка может быть полезна и для того, чтобы разобраться в алгоритме

Отладка

Отладка — этап разработки компьютерной программы, на котором обнаруживают, локализуют и устраняют ошибки.

Отладчик (дебаггер, debugger) — компьютерная программа, предназначенная для поиска ошибок в других программах. Отладчик позволяет сделать трассировку.

¹трассировка может быть полезна и для того, чтобы разобраться в алгоритме

Отладка

Отладка — этап разработки компьютерной программы, на котором обнаруживают, локализуют и устраняют ошибки.

Отладчик (дебаггер, debugger) — компьютерная программа, предназначенная для поиска ошибок в других программах. Отладчик позволяет сделать трассировку.

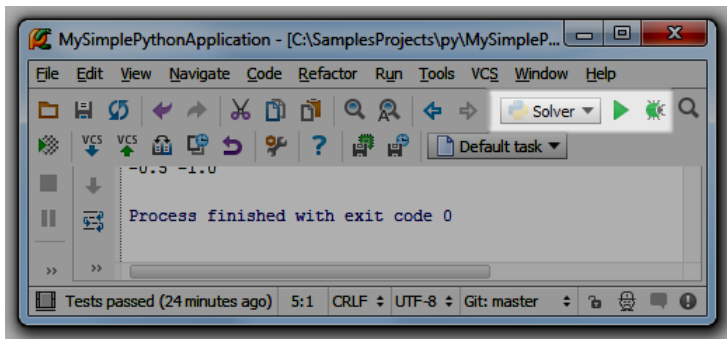
Трассировка — процесс пошагового (оператор за оператором) выполнения программы¹.

¹трассировка может быть полезна и для того, чтобы разобраться в алгоритме

Отладка

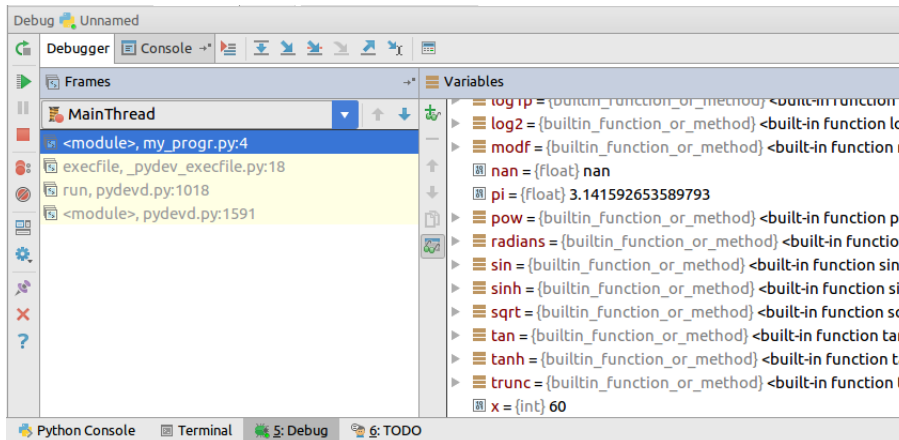
Как правило транслятор (интерпретатор или компилятор) языка программирования поставляются вместе с отладчиком.

Среда программирования, в свою очередь, предоставляет как правило два варианта запуска программы: обычный и с отладчиком.



Отладка

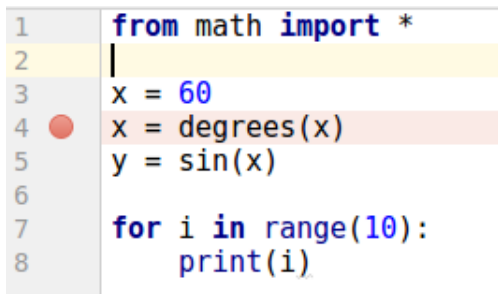
Запуск программы с отладчиком (отладка) обычно работает медленнее, но позволяет приостановить выполнение программы на любой инструкции и просмотреть её *состояние*.



Отладка

Чтобы остановить выполнение программы на определённой строчке (без её выполнения) нужно заранее создать там **точку останова (brakepoint)**.

Как правило в IDE точки останова создаётся кликом возле номера строки.



```
1 from math import *
2 |
3 x = 60
4 x = degrees(x)
5 y = sin(x)
6
7 for i in range(10):
8     print(i)
```

The image shows a code editor with a Python script. A red circle, representing a breakpoint, is placed next to line 4, which contains the code `x = degrees(x)`. The code is as follows:

Отладка

Во время отладки возможны различные режимы выполнения программы:

- ▶ Выполнение до следующей точки останова
- ▶ Выполнение одной строки кода
- ▶ Выполнение до курсора

Демонстрация

2

Outline

Предыдущие темы

IDE

Jupyter

Wing

Отладка

Циклы

while

вычисление сумм и произведений

for

Операторы управления циклом

Составные типы данных

Списки

Кроме условного выполнения операторов (с помощью `if .. else`) часто возникает необходимость повторения операторов заданное число раз или до достижения необходимого эффекта.

Пример...

- ▶ найти сумму (произведение) нескольких членов ряда
- ▶ обработать набор данных (например чисел) применив ко всем одинаковые операции
- ▶ повторять запрос на ввод данных пользователя до тех пор пока эти данные не будут введены корректно
- ▶ ...

Цикл

Цикл — разновидность управляющей конструкции в высокоуровневых языках программирования, предназначенная для организации многократного исполнения набора инструкций.



Цикл

Последовательность инструкций, предназначенная для многократного исполнения, называется **телом цикла**.

Единичное выполнение тела цикла называется **итерацией**.

Циклы

Циклы в Python

- ▶ Цикл с предусловием. `while`
- ▶ Цикл со счётчиком (совместный цикл). `for`

Виды циклов

- ▶ Цикл с предусловием
- ▶ Цикл с постусловием
- ▶ Цикл со счётчиком
- ▶ Совместный цикл

Язык программирования не обязательно содержит операторы всех перечисленных циклов.

К тому же, некоторые виды циклов можно реализовать на языке программирования с помощью комбинации условного, циклического оператора, а также операторов досрочного выхода из цикла и пропуска итерации.

Outline

Предыдущие темы

IDE

Jupyter

Wing

Отладка

Циклы

while

вычисление сумм и произведений

for

Операторы управления циклом

Составные типы данных

Списки

Циклы. While

Цикл с предусловием. Короткая форма

```
while условие:      # заголовок цикла  
    оператор1        # тело цикла  
    оператор2        # тело цикла  
    ...              # тело цикла  
  
    # тело цикла заканчивается там,  
    # где закончивается вложенный блок (отступы)
```

Тело цикла (оператор1, оператор2, ...) будет выполняться до тех пор пока *условие* истинно.

Циклы. While

Цикл будет завершён, когда условие станет ложным.

Условие выхода из цикла - это ситуация, когда условие в цикле становится ложным.

Чтобы цикл завершился, тело цикла должно в конечном итоге повлиять на условие. Условие в конечном итоге должно стать ложным.

Циклы. While. Пример

```
i = 5
while i > 0:
    print(i, "bottles of beer on the wall" )
    i = i - 1
```

Что будет напечатано?³

³Работу цикла легко проследить запустив отладчик и выполнить строчку за строчкой

Циклы. While. Пример

```
i = 5
while i > 0:
    print(i, "bottles of beer on the wall" )
    i = i - 1
```

Что будет напечатано?³

```
5 bottles of beer on the wall
4 bottles of beer on the wall
3 bottles of beer on the wall
2 bottles of beer on the wall
1 bottles of beer on the wall
```

³Работу цикла легко проследить запустив отладчик и выполнить строчку за строчкой

Циклы. While

Если условие изначально ложно, то тело цикла не выполнится ни разу, будут сразу выполняться те операторы которые находятся после цикла.

```
while False:
    print("Это сообщение не будет напечатно")
    print("И это тоже")

print("А это увидят все")
```

Циклы. While

Если условие *всегда* истинно, то тело цикла будет выполняться *бесконечно*.

```
# этот цикл будет работать бесконечно  
while True:  
    print("Я буду учить Python!")
```

```
# строка ниже никогда не выполнится  
print("И буду знать его в совершенстве!")
```

Такие циклы, без условия выхода называют **бесконечными циклами**

Циклы. While

Цикл с предусловием. Полная форма

```
while условие :  
    оператор1  
    оператор2  
    ...  
else:  
    операторN1  
    операторN2
```

операторN1 и операторN2 выполнятся когда условие станет ложным.

Outline

Предыдущие темы

IDE

Jupyter

Wing

Отладка

Циклы

while

вычисление сумм и произведений

for

Операторы управления циклом

Составные типы данных

Списки

Циклы. While

Примеры

Найти сумму ряда $\sum_{i=1}^n \frac{1}{i} = \frac{1}{1} + \frac{1}{2} + \dots + \frac{1}{n}$

```
n = 5
s = 0 # для хранения (накопления) суммы
i = 1 # счётчик цикла

while i <= n:
    s = s + 1/i
    i = i + 1

print("сумма",n,"членов ряда: ", s)

сумма 5 членов ряда: 2.2833333333333333
```

Циклы. While

Примеры

Пример. Вычисление суммы ряда $\sum_{i=1}^n \frac{1}{i} = \frac{1}{1} + \frac{1}{2} + \dots + \frac{1}{n}$ с заданой точностью ϵ^4

```
eps = 0.2
a = 1 # сразу вычислим первое слагаемое
s = 0
i = 1 # счётчик цикла

print("Члены ряда:")
while a > eps: # вычисляем сумму, только если слагаемое больше eps
    s = s + a
    i = i + 1 # следующий номер слагаемого
    a = 1 / i # следующее слагаемое

print()
print("сумма ", i-1, "членов ряда: ", s)
```

~~сумма 5 членов ряда: 2.283333333333333~~

⁴найти сумму всех слагаемых, которые больше ϵ

Циклы. While

Примеры

Вычисление факториала числа N

```
N = 6
i = N
F = 1  # используем переменную как "накопитель"
while i != 1:  # != означает "не равно"
    F = F * i
    i = i - 1

print(N, "! =", F)
```

6 ! = 720

Outline

Предыдущие темы

IDE

Jupyter

Wing

Отладка

Циклы

while

вычисление сумм и произведений

for

Операторы управления циклом

Составные типы данных

Списки

Циклы. for

Цикл for используют когда известно число итераций ⁵

```
for перемен in последовательность : # заголовок цикла  
    оператор1 # тело цикла  
    оператор2  
    ...
```

перемен - **переменная цикла**.

Переменная цикла for не требует дополнительного объявления вне цикла.

Оператор for будет записывать в переменную цикла один элемент последовательности, затем выполнять тело цикла; записывать в переменную цикла следующий элемент последовательности, выполнять тело цикла и так далее пока в последовательности не закончатся элементы.

⁵задаётся числом элементов в последовательности

Циклы. for

Для генерирования последовательности чисел часто используется функция range

Повторить тело цикла n раз

```
for i in range(1, n+1 ):    # заголовок цикла
    оператор1                # тело цикла
    оператор2
    ...
```

i - переменная цикла.

Правила выбора имени переменной цикла - такие же, как и для остальных переменных.

Циклы. функция range

С помощью функции range создаётся последовательность, которая задаёт значения переменной цикла⁶:

```
range( start, finish + 1)
```

start - начальное значение

finish - конечное значение

Причём range отнимает от конечного значения 1, поэтому в примере конечное значение задано как $n + 1$

⁶но range можно использовать не только для этого

Циклы

Функция `range` не создаёт все элементы последовательности сразу (для экономии памяти), а выдаёт значения по одному, например когда нужно записать в переменную цикла

Просмотреть последовательность целиком можно в интерактивном режиме python используя функцию `list`:

```
list( range(1, 5) )  
[1, 2, 3, 4]
```

```
list( range(10, 20) )  
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```


Циклы

Функцию `range` можно вызывать и с одним параметром, тогда последовательность будет генерироваться от 0 до параметр - 1

```
range(5)      # 0, 1, 2, 3, 4
```

```
range(0, 5)   # 0, 1, 2, 3, 4
```

```
range(3,7)    # 3, 4, 5, 6
```

```
range(-3, 1)  # -3, -2, -1, 0
```

Циклы

`range(a, b, d)` - создаёт последовательность от **a** до **b** с шагом **d**

Последнее число никогда не включается в последовательность.

```
range(2,10,2)  # 2, 4, 6, 8  
range(5,0,-1)  # 5, 4, 3, 2, 1  
range(0,25,5)  # 0, 5, 10, 15, 20
```

Циклы. for

Число итераций цикла определяется разностью начального и конечного значений + 1:

число итераций = finish - start + 1

```
for i in range( 1, n+1 ):    # заголовок цикла
    print("i =", i)
```

```
# вывод на экран
```

```
i = 1
```

```
i = 2
```

```
i = 3
```

```
i = 4
```

```
i = 5
```

Циклы. for

Пример

Цикл for хорошо подходит для вычисления суммы конечного числа слагаемых ряда (лучше чем while)

Найти сумму ряда $\sum_{i=1}^n \frac{1}{x^2}$

```
n = 5
s = 0  # переменная для суммы

for i in range( 1, n+1 ):    # заголовок цикла
    s = s + 1 / i**2

print( "Сумма", n, "слагаемых ряда: ", s)
```

Сумма 5 слагаемых ряда: 1.4636111111111112

Циклы

Иногда для анализа работы цикла бывает полезным разобрать его выполнение по шагам записывая значение переменных, что изменяется в цикле

```
j = 0
k = 0
for i in range(6):
    j = j + i
    k = k + j
```

i	j	k
0	0	0
1	0	0
2	1	1
3	3	4
4	6	10
5	10	20

Цикл for

Напечатать все числа из списка:

```
l = [1, 40, -4, 3.14, 0]
for e in l :
    print(e)
```

Что будет напечатано:

1
40
-4
3.14
0

Цикл for

Вышеприведённый вариант (без использования индексации) всегда следует предпочитать варианту с индексацией

```
l = [1, 40, -4, 3.14, 0]
for i in range(len(l)) :
    print(l[i])
```

При прочих равных стоит выбирать то решение, которое короче и не требует дополнительных переменных.

Цикл for

Замечание: Последовательность можно вывести на экран и функцией `print()`:

```
print( [1, 40, -4, 3.14, 0] )
```

Однако, такие последовательности всегда выводятся вместе со скобками, что может ухудшить восприятие информации:

```
[1, 40, -4, 3.14, 0]
```


Цикл for

Переменную цикла можно использовать в выражениях
Напечатать степени двойки:

```
for i in [0, 1, 2, 3, 4, 5] :  
    print(2**i)
```

Что будет напечатано:

1
2
4
8
16
32

Взаимозаменяемость циклов `for` и `while`

- ▶ Цикл `while` универсален.
- ▶ `for` всегда можно заменить на `while`
- ▶ однако порой `for` позволяет решить задачу меньшим количеством кода
- ▶ кроме того, использование `for` иногда даёт выигрыш в производительности.

Например часто используемая совместно с циклом `for` функция `range` создаёт последовательность не целиком, а по одному элементу. Эта функция выдаёт следующий элемент в тот момент когда его нужно записать в переменную цикла.

Когда использовать for, а когда while?

- ▶ for подойдёт когда известно число итераций цикла
Например, число элементов в последовательности
- ▶ while можно использовать во всех остальных случаях

Outline

Предыдущие темы

IDE

Jupyter

Wing

Отладка

Циклы

while

вычисление сумм и произведений

for

Операторы управления циклом

Составные типы данных

Списки

Операторы управления циклом

break - прерывает выполнение тела цикла и производит безусловное завершение всего цикла.

Т.е. это оператор безусловного выхода из цикла.

Любые операторы описанные в теле цикла после **break** не выполняются.

```
while условие :  
    оператор1  
    оператор2  
    break  
    оператор3
```

Оператор3 никогда не выполнится. Цикл завершится на первой итерации
break завершит как цикл **while** так и **for**.

Операторы управления циклом

Оператор **break** полезно использовать только совместно с оператором ветвления **if**.

Поиск индекса элемента в списке

```
i = 0
l = [3, 14, 15, 9, 26, 5]
x = 9  # искомое значение
while i < len(l):
    if l[i] == x:
        break
    else:
        i = i + 1
if i == len(l):
    print("Значение ", x, " не найдено")
else:
    print("Значение ' ", x, "' находится под порядковым номером
```

Значение ' 9 ' находится под порядковым номером 3

Бесконечные циклы

Оператор **break** позволяет спользовать "бесконечные"циклы:

```
while True :  
    оператор1
```

Пример.

Запрашивать у пользователя число натуральное число.

```
print("Введите натуральное число")  
while True :  
    n = int( input( "n = " ) )  
    if n > 0:  
        break  
    else:  
        print("Введите НАТУРАЛЬНОЕ число")
```

Операторы управления циклом

continue - прерывает выполнение текущей итерации цикла и переходит на следующую.

```
while условие :  
    оператор1  
    оператор2  
    continue  
    оператор3
```

Оператор3 никогда не выполнится. Цикл завершится

"естественным образом когда условие в заголовке не выполнится.

Операторы управления циклом

Найти действительные квадратные корни элементов последовательности

```
l = [2, 9, 3, -8, -3, -10, 10]
for x in l:
    if x < 0:
        continue
    print("sqrt(",x,") = ",sqrt(x))
```

sqrt(2) = 1.4142135623730951

sqrt(9) = 3.0

sqrt(3) = 1.7320508075688772

sqrt(10) = 3.1622776601683795

Операторы управления циклом

continue и **break** - избыточны. Любой цикл можно организовать без этих операторов. Как обойтись без **continue**?

Операторы управления циклом

continue и **break** - избыточны. Любой цикл можно

организовать без этих операторов. Как обойтись без continue?

```
for x in l:
    # условие завершения итерации поменялось
    # на условие её проложения
    if x >= 0:
        print("sqrt(",x,") = ",sqrt(x))
```

Операторы управления циклом

Как обойтись без break?

Замечание об объявлении переменной внутри блока

```
if условие :  
    x = 10  
else:  
    y = 20  
  
print(x)  
print(y)
```

В чём проблема?

Замечание об объявлении переменной внутри блока

```
if условие :  
    x = 10  
else:  
    y = 20  
  
print(x)  
print(y)
```

В чём проблема? Переменная `x` будет объявлена только если условие выполнится.

Переменная `y` будет объявлена только если условие **не** выполнится

Однако значения этих переменных выводятся безусловно, вне зависимости от этого какая переменная была объявлена.

Как решить эту проблему?

Замечание об области видимости переменной

```
x,y = None, None
if условие :
    x = 10
else:
    y = 20

print(x)
print(y)
```

Проблема с областью видимости возникает, если переменная объявляется внутри любого блока, который может и не выполняться. Например в теле цикла.

Outline

Предыдущие темы

IDE

Jupyter

Wing

Отладка

Циклы

while

вычисление сумм и произведений

for

Операторы управления циклом

Составные типы данных

Списки

Составные типы

В программах часть приходится хранить не только отдельные значения, но и целые наборы значений. Причём число таких элементов не всегда может быть известно.

Python для хранения наборов значений используются три категории типов:

- ▶ Последовательности (упорядоченный набор значений)
 - ▶ Строки (str)
 - ▶ Списки (list)
 - ▶ Кортежи (tuple)
- ▶ Множества (set)
- ▶ Отображения (dict)

Outline

Предыдущие темы

IDE

Jupyter

Wing

Отладка

Циклы

while

вычисление сумм и произведений

for

Операторы управления циклом

Составные типы данных

Списки

В Python списки (list) на самом деле динамические массивы⁷
Но со этими ненастоящими списками можно работать во многом как со списками. Объявить пустой список:

```
l = []
```

Можно сразу задать значения:

```
l = [ 1, 2, 3 ]
```

⁷<https://docs.python.org/3/faq/design.html#how-are-lists-implemented-in-python>

Добавление элементов Списку не нужно задавать начальный размер потому, что в любой момент можно добавить ещё один элемент в конец.

[42] - это список из одного элемента

```
l = l + [42]    # l = [1, 2, 3, 42]
```

или

```
l.append(42)    # l = [1, 2, 3, 42]
```

Списки

Элементами списка могут быть практически любые типы

```
l = [ 10, "Python", 3.14, True ]
```

В том числе другие списки

```
l2 = [ 1, [0,1,2,3] ]
```

Уровень вложенности может быть любым Например можно организовать древовидную структуру

```
tree = [ [[1], 2, [3]], 4, [[5], 6, [7]] ]
```

Как можно представить такой список?

Некоторые операции со списками

```
l = [10, 20, 30, 42]
```

```
n = len(l)  # получить длину списка
```

```
x = l[2]    # доступ к элементам. индексация с 0  
# x = 30
```

```
z = l[-1]   # доступ к последнему элементу
```

```
l2 = [0] * 128  # создание списка из 128 нулей
```

```
l2 = [0] * 64 + [1] * 64  # список из 64 нулей и 64 единиц
```

```
l3 = [0, 1] * 64  # список из чередующихся нулей и единиц
```

Списки

Создание списка из последовательности целых чисел

```
l = list( range(0,10) )
```

```
l = [0,1,2,3,4,5,6,7,8,9]
```

```
l = list( range(0,10,2) )
```

```
l = [0,2,4,6,8]
```

```
l = list( range(10,0,-1) )
```

```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

```
l = list( range(25,0,-5) )
```

```
[25, 20, 15, 10, 5]
```

Списки

Создать список из n случайных чисел

```
from random import random
# функция random возвращает псевдослучайное число
# в интервале от 0 до 1
```

Традиционный способ

```
n = 100
l = []
for i in range(n):
    l = l + [random()]
```


Быстрее и короче. С помощью генератора списка.

```
n = 100  
l = [random() for i in range(n)]
```

Списки

Проход по списку.

"Традиционный способ"

```
l = [1,2,3,4]

for i in len(l):
    print(l[i])
```

С помощью совместного цикла

```
l = [1,2,3,4]

for e in l:
    print(e)  # в теле такого цикла нельзя менять e
```

Если нужно только перебрать элементы одной последовательности то всегда следует использовать перебор безиндексов.

Такой код лаконичнее и лучше воспринимается.

К тому же последовательное обращение к элементам списка просиходит быстрее, чем обращение по номеру элемента.

```
for e in l:  
    print(e)    # в теле такого цикла нельзя менять e
```

Доступ к элементам и срезы

```
a = [1, 3, 8, 7]
```

К элементам можно обращаться по их номеру от конца списка

```
a[-1] # последний, 7
```

#

```
a[-4] # 1
```

Если указан номер несуществующего элемента, то происходит ошибка времени выполнения

```
a[20]
```

IndexError: list index out of range

Доступ к элементам и срезы

Срезы

a[от : до : шаг]

от - если не указан, то от начала списка

до - не включается в срез. по умолчанию - конец списка

шаг - по умолчанию равен 1

a[:]

Весь список. от первого элемента до последнего (включая последний) с шагом 1

a[::]

Весь список. от первого элемента до последнего (включая последний) с шагом 1

3, 8, 7

Доступ к элементам и срезы

```
>>> a[1:]  # со второго элемента до последнего  
[3, 8, 7]  
>>> a[:3]  # с первого элемента до третьего  
[1, 3, 8]  
>>> a[::2] # с первого и до последнего, с шагом 2  
[1, 8]
```

Срезы

```
>>> a = [1, 3, 8, 7]
```

Как получить элементы списка в обратном порядке?

Срезы

```
>>> a = [1, 3, 8, 7]
```

Как получить элементы списка в обратном порядке?

```
>>> a[::-1]
```

```
[7, 8, 3, 1]
```

```
>>> a[:-2]
```

```
[1, 3]
```

```
>>> a[-2::-1]
```

```
[8, 3, 1]
```

```
>>> a[1:4:-1]
```

```
[]
```


Двумерный список

```
m = [ [1,2,3], [4,5,6], [7,8,9] ]  
for l in m:  
    for e in l:  
        print(e, end="")  
    print()
```

доступ по индексу

```
m[1][2] # 5
```

Двумерный список

Создание двумерного списка (матрицы) $n \times m$

n - число строк

m - число столбцов

```
A = []  
for i in range(n): # цикл по строкам матрицы  
    A = A + [ [0]*m ]
```

Ссылки и литература

- ▶ Как использовать Jupyter (ipython-notebook) на 100%
Доклад о Jupyter

Основная литература

- ▶ Федоров, Д. Ю. Программирование на языке высокого уровня python : учебное пособие для прикладного бакалавриата Содержит краткое описание языка.
- ▶ ru.wikibooks.org/wiki/Python - Викиучебник
- ▶ Лутц М. Изучаем Python. 2010. - 1280 с. Содержит подробное описание языка.
- ▶ Официальная документация Python3
`help(имя)`

Ссылки и литература

Ссылка на презентации github.com/VetrovSV/Programming